Applying Software Product-Line Architecture

Software product-line architecture is a powerful way to control the risks and take advantage of the opportunities of complex customer requirements, business constraints, and technology, but its success depends on more than technical excellence.

David Dikel
Applied Expertise Inc.

David Kane
Applied Expertise Inc.

Steve Ornburn
Nortel
(Northern Telecom)

William Loftus
WPL Laboratories Inc.

Jim Wilson
Applied Expertise Inc.

Many organizations today are investing in software product-line architecture—for good reason: A well-executed architecture enables organizations to respond quickly to a redefined mission or to new and changing markets. It allows them to accelerate the introduction of new products and improve their quality, to reengineer legacy systems, and to manage and enhance the many product variations needed for international markets.

Software architecture is the structure of the components of a program or system, their interrelationships, and the principles and guidelines governing their design and evolution.[1] Sharing a common software architecture across a product line brings a core set of knowledge and assets to the development process. Product-line architecture not only reduces the complexity and cost of developing and maintaining code, but also streamlines the production of documentation, training materials, and product literature.

However, technically excellent product-line architectures do fail, often because they are not effectively used. Some are developed but never used; others lose value as product teams stop sharing the common architecture; still others achieve initial success but fail to keep up with a rapidly growing product mix. Sometimes the architecture deterioration is not noticed at first, masked by what appears to be a productivity increase.

To learn what factors determine the effective use of software architecture, we looked at Nortel (Northern Telecom), a company with nearly 20 years of experience developing complex software architecture for telecommunications product families. We identified six principles that help reduce the complexity of an evolving family of products and that support and maintain the effective use and integrity of the architecture. Nortel's experience provides compelling evidence of software architecture's benefits and risks and shows the importance of organizational behaviors.

THE NORTEL STUDY

Nortel's leadership in digital switch architecture enabled the company to capture US and international markets. After the 1984 breakup of AT&T, only Nortel could provide Bell operating companies with the capability to offer customers equal access to long-distance carriers. The company's high product quality allowed it to become the first digital switch supplier in Japan.

After nearly 20 years of successful use, however, the digital switch architecture began to show signs of needing renewal. In the late 1980s, the company considered a major restructuring of the architecture, but the CEO decided to wait. This decision had severe consequences, with a reduction in product quality and a tripling in the length of release cycles. Nortel later attributed these problems to architecture breakdown.[2] The company took action and rebounded. After reporting substantial losses in 1993, Nortel posted profits of $408 million in 1994, $473 million in 1995, and $623 million in 1996.

In our study we realized that technical factors—the focus of most research—do not by themselves explain the success of a product-line architecture. Only in conjunction with appropriate organizational behaviors can software architecture effectively control project complexity. These behaviors have been documented in studies of software reuse[3,4] and in systems architecture,[5] but our purpose was to identify these behaviors for software architecture in a long-term effort to build large-scale, mission-critical systems—hence our choice of Nortel.

Although our effort was an exploratory case study, we followed a rigorous methodology.[6] Our first step was to assemble a group of 16 advisers with broad software architecture expertise, including leading figures like Grady Booch and Robert

A well-executed architecture enables organizations to respond quickly to a redefined mission or to new and changing markets.

Charette and representatives from such organizations as Hewlett-Packard, IBM, NASA, the Software Engineering Institute, Texas Instruments, and the US Navy. These advisers, especially Booch, helped us develop a set of six organizational principles believed critical to the long-term success of a software architecture:

- Focusing on simplification, minimization, and clarification.
- Adapting the architecture to future customer needs, technology, competition, and business goals.
- Establishing a consistent and pervasive *architectural rhythm*—regular architecture and product releases that help coordinate the actions and expectations of all parties.
- Partnering and broadening relations with stakeholders.
- Maintaining a clear architecture vision across the enterprise.
- Proactively managing risks and opportunities.

Our next step was to conduct our research and see how these principles fit Nortel's experience. We studied the business unit at Nortel that builds a family of digital loop carriers (DLCs)—products that provide an intermediate connection between central office switches and end phone systems. Over several years, the 80-person DLC group had successfully managed architectural complexity through a series of software releases. In fact, the group had been able to reduce cycle time by 45 percent, still maintaining high quality.

The DLC group's products share a common software architecture and hardware platform. In the two releases it makes each year, the group adds or changes 3 to 5 percent of the software's 4 million lines of code. The architecture was constructed in several layers with large-grained components, some of which had been developed by other Nortel organizations for other telecommunications products. A subteam managed formal customer-supplier relationships with the groups responsible for this reused software.

We visited the Nortel facility twice, collecting information through interviews, documentation reviews, quantitative data analysis, and observation. The interviewees ranged from engineers to directors, and together they had hands-on experience with 95 percent of the architecture's components. Also, many of the interviewees had worked with architectures in other departments and were able to put the DLC group's work into perspective.

In fact, a number of the engineers we interviewed had had long-term involvement with Nortel's digital switch architecture. The collapse and revival of that architecture profoundly affected them, and they continued to apply the lessons learned then to their present work with the DLC group. One example is the practice of *cloning*—the duplication of an architectural component and its maintenance by a separate owner—which is seen as both a symptom and a cause of architecture deterioration and is now frowned upon.

While cloning had offered a means to quickly develop new features, it often had far-reaching consequences. Duplication of code greatly complicated product tracking and management. It also dramatically increased the maintenance burden of each product and reduced the possibility of reusing externally maintained components. Even with a world-class configuration management toolset and process, Nortel was unable to halt the deterioration of the architecture without first addressing organizational issues.

After gathering our information, we classified and analyzed it according to the principles. In this way, we were able to verify that we had answered all our questions and could thus see how each principle applied to the DLC group's experiences. We also incorporated data we had collected that compared the organization's performance with industry norms and with the organization's historical performance.

Finally, we held a workshop for the advisers where we presented our results. This gave them a chance to confirm our findings and add perspectives of their own. Then, after a review by Nortel, we finalized our findings.

THE PRINCIPLES

In our collection and analysis of the data, we asked certain questions of each principle: Is it critical? Is it actually done? If so, how? In what context? How does it contribute to the group's success? How does the principle relate to other principles? How does it affect complexity?

We realized that the principles affect not only architecture, but also system complexity and its negative consequences. Two principles—focusing on simplification and adapting for the future—directly address complexity, but we found evidence that to one degree or another all six principles affect system complexity. In fact, we were surprised by the influence that some—such as establishing a rhythm and partnering with stakeholders—had on complexity.

In the DLC group, we found that some principles were applied informally. By design, management kept documentation illustrating the principles to a minimum. While this approach has proven to be effective, it requires monitoring and is vulnerable to changes in the organization or its customer base.

Focusing on simplification

One study adviser, Booch, has written that "a ruthless focus on simplification, clarification, and minimization" is essential for a successful large software

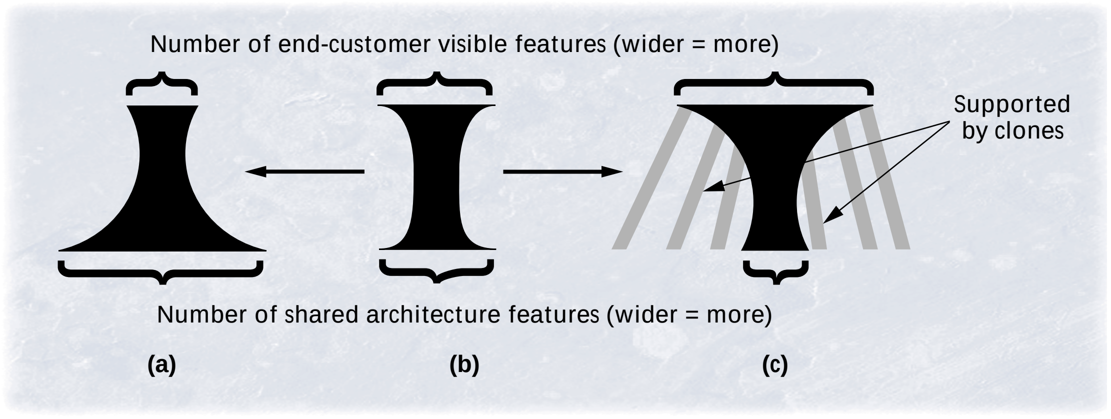
Number of end-customer visible features (wider = more)
Supported by clones
Number of shared architecture features (wider = more)
(a)
(b)
(c)

Figure 1. Simplification requires balancing the tension between the needs of current and potential users. (a) An architecture overly focused on future needs; (b) a well-balanced architecture; (c) an architecture overly focused on immediate needs.

project.[7] We found evidence that the DLC group strove to define minimal characteristics and build them into the core elements of the architecture. Engineers were encouraged to simplify existing and legacy code, being rewarded with recognition, pay increases, and promotions. Managers and component owners monitored churn (the addition and modification of code) and the complexity of interfaces, keeping lists of areas for possible simplification. We found, however, that simplification did require a compelling business case based on the maintenance cost and risk of failure versus the effort and risk to simplify the component—as well as someone to lead the effort.

There were several ways in which complexity grew. Figure 1 illustrates how a simplification effort, when focused on a single driver, can be applied too much or too little. The figure shows the results of two opposing drivers—delivering features visible to end users now or building and maintaining shared architecture features for the future—and their ideal balance. Figure 1a illustrates the situation in which a group tries to create an architecture that is too flexible—one that is intended for a wide variety of loosely defined uses. The resulting components will be too general, too full of unneeded features, and too slow to be useful. Engineers would then bypass these components, introducing more specialized and more efficient alternatives. Figure 1c illustrates the opposite situation, when a group tries to quickly fashion point solutions for a variety of customers, which results in a tendency to clone large chunks of code. By focusing on simplicity, however, the DLC group found a sensible and profitable balance, an ideal illustrated in Figure 1b.

But how do you find a balance? There is certainly no point in simplifying by chopping away the most critical feature. A key to the right answer lies in Booch's phrase: the solution must be clear and minimal as well as simple, a process that is illustrated in Figure 2. When simplification focused on delivering customer value and achieving business goals, the chances for success increased. Another key is to ensure that each of the remaining five principles are applied. For example, focus on partnering should ensure that you don't get rid of a feature that is critical to a partner's business unit.

We did observe limits to simplification. As supported by W. Ross Ashby's concept of *requisite variety,*[8] there is an appropriate level of complexity for software architecture and its supporting processes. Requisite variety suggests that a system should be as complex as its environment. If the software architecture becomes more complex than its environment, it may become too expensive for the organization to support. If some facets of the software architecture

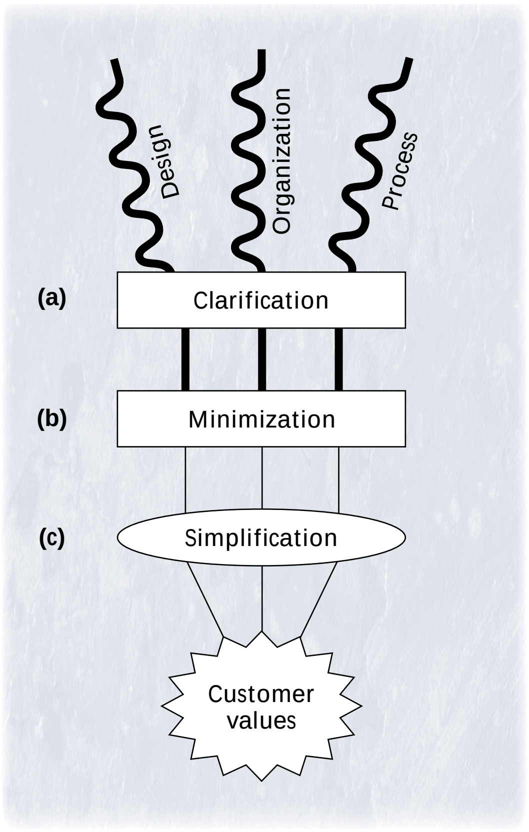

Figure 2. Chances of success increase when developers (a) clarify the problem they are trying to solve and then (b) minimize and (c) simplify the solution by focusing on delivering customer value and achieving business goals.

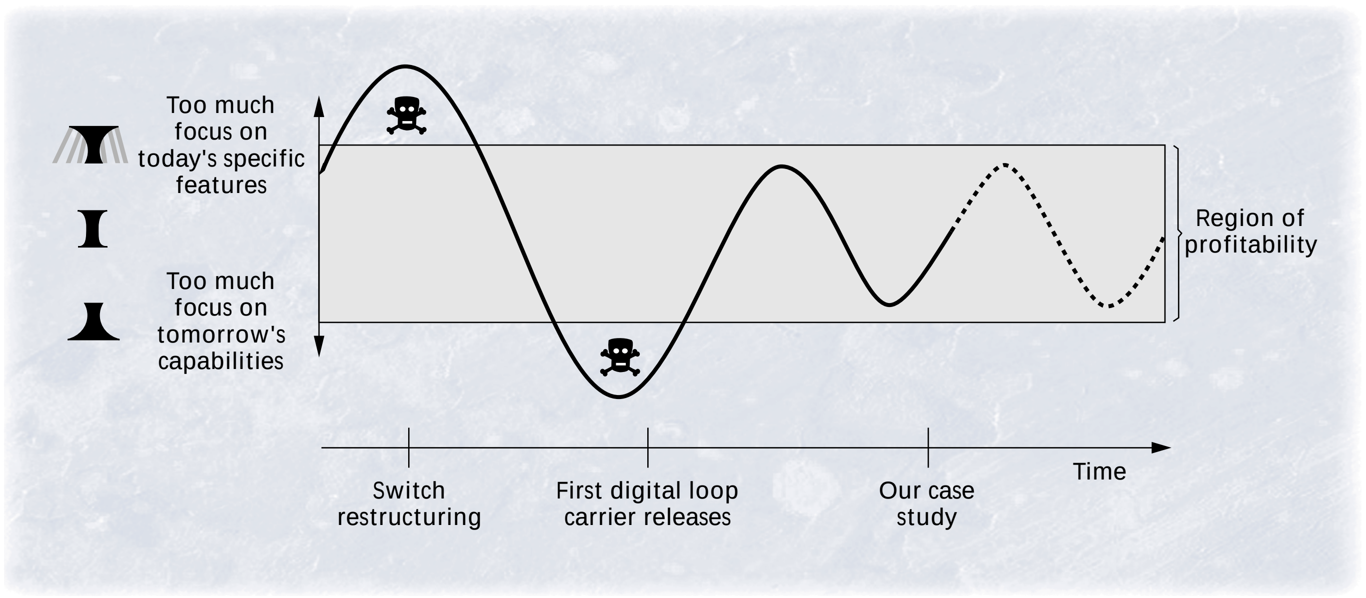
Too much focus on today's specific features
Too much focus on tomorrow's capabilities
Region of profitability
Switch restructuring
First digital loop carrier releases
Our case study
Time

Figure 3. In the case of architecture, balancing short-term results and far-reaching vision is a lot harder than it looks. Rhythm helps maintain a profitable balance.

are oversimplified, then the organization may not be able to respond to a diverse set of customers or to marketplace dynamics.

Adapting to future needs

In the course of the study, we were not able to isolate many predictive activities such as forecasting market and technology trends that were specific to software architecture. However, the group did show that it had learned how to balance architecture development between current customer needs and anticipated future needs. Sometime earlier, another Nortel business unit had set aside 10 percent of its R&D funding to build new technology and add capacity to the architecture, but business pressures in 1986 caused the shifting of these funds to other uses. The unit's vision for its architecture deteriorated, and by the late 1980s developers would clone rather than share architecture components, increasing the amount of code. Initially this was seen as an increase in productivity,[9] but by 1993 the time required to add features had tripled.[2]

The DLC group had also learned that it is possible to put too much emphasis on the future. When the project originally started, engineers with fresh memories of the digital switch collapse pushed for an architecture with broad capability. This proved burdensome, with the product line taking too long to develop, missing its market window. The results were consistent with Robert Cooper's analysis of successful product introduction, which shows that sufficient product value needs to be delivered to customers early enough to establish initial product success.[10]

The DLC group learned from both experiences, keeping its code growth steady and its architecture clear. When the DLC group balanced future needs with current needs, it was able to successfully add large-scale functionality, to enter new markets, and to increase product sales. One manager said that instead of creating feature-rich products, they were creating products with "just the features that make you rich." The group planned the evolution of the architecture, tying new features to scheduled releases over the course of two years. Also, engineers worked collaboratively with customers and thus understood not only the technical aspects of the features, but also their market implications.

Establishing architectural rhythm

We were surprised how a consistent and pervasive *architectural rhythm*—regular releases that help coordinate actions and expectations—dramatically simplified the process for all players. Figure 3 illustrates how rhythm relates to the trade-offs made between function and capability described earlier.

Rhythm ensures that all involved—including customers, engineers, suppliers, managers, and executives—understand important issues and know their own responsibilities.

In the processes of the DLC group, we found daily, weekly, monthly, and lengthier rhythms. Planning, development, testing, problem resolution, creation of documentation, issuing of releases, and marketing all had their own predictable rhythm, and this allowed for synchronization and closure of tasks. Everyone knew the milestones preceding a release and the regular intervals between them.

It was not always that way in the DLC group. Initially managers did not place priority on completing features on schedule or on weekly builds, and subsequently the first release was delayed by several years. An internal review pointed out the error in this, and a rhythm was established—something one senior manager considered "the best thing we did." The product later had successful sales growth and high customer ratings.

Our interviewees pointed out too that there was a period when some managers were less rigid about maintaining schedules, allowing feature development to proceed as "managed exceptions" to the rhythm. This increased the number of releases per year and greatly complicated management of the development process, raising the possibility that important factors would be overlooked. For example, unexpected changes in content would invalidate earlier architectural decisions, requiring last-minute changes in the design of shared components. With increased schedule pressure, many of these changes were "hacked" rather than carefully integrated into the component's design. Finally, Nortel enforced rhythm and corrected the problem.

Partnering with stakeholders

We added partnering as a critical principle when our HP advisers told us that their group makes development of partnering skills a first step when an internal organization requests product-line software reuse. When we studied the DLC group, we saw that partnering was a key element in delivering the value of Nortel's architecture. We uncovered examples of partnering with internal stakeholders, like users and developers of architectures, and we saw that both official policy and unwritten rules encouraged partnering and therefore the reduction of architecture and product complexity.

Several of the engineers told us that when they or other engineers had unilaterally changed a component and broken the weekly compile, they raised the ire of their bosses and co-workers. We also heard about component owners who would not respond to the needs of the users and were therefore sanctioned. Now, when users wanted changes to a component, they negotiated directly with the component owner. As a result, weekly compiles rarely broke—even when a wide range of individuals in various locations had the ability to change an architectural component. Crashes remained minimal even when components were modified to meet the requirements of new products.

Partnering also influences promotions and raises. Many of the senior engineers, architects, and managers told us stories of how, earlier in their careers, when faced with a new requirement, they identified other customers and crafted the component to address the needs of more than one group. Once the component achieved its dual purpose, the organization provided additional resources to maintain the component and invited the component's champion to be a key member of the design team.

Engineers, too, were rewarded for productive use of their personal networks and spheres of influence. Managers found that those with effective networks were better able to negotiate changes, and there was less management involvement and less cloning. Engineers, too, were alert to business issues and trade-offs. They also had a consistent understanding of how and when to elevate to management conflicts between component owners and engineers wanting changes.

Maintaining vision

Without a clear vision, something as abstract as a software architecture cannot be effectively shared. Developers and users must feel confident that they know the general purpose of the designs and code and that they can identify individuals who know the details. Without a clear vision, software architecture is meaningless and incapable of supporting a product line. For example, a clear vision was first needed before DLC engineers could request and negotiate changes with component owners. All parties knew the ultimate objectives and who was responsible for what.

In fact, we found that members of the DLC group drew similar high-level representations of the architecture, and strongly identified components with the individuals responsible for them. One manager said: "You could create a chart with the faces of the owners of all the major architecture components, and the staff could fill in the names of components based upon the faces." Thus, the structure of the architecture paralleled the structure of the organization. This, we believe, made the architecture more tangible and helped maintain its integrity.[5] This is not to say that the organization's structure was allowed to stagnate. Engineers were regularly rotated, not only to handle new responsibilities, but also to transfer technical depth and breadth and to increase understanding of the entire architecture.

The DLC group learned from other Nortel experiences about the need for a shared architectural vision. With market growth in the mid-1980s, the switch architecture group hired many new engineers, but did not sufficiently teach them the organization's vision. Many did not even know who to go to about specific components, and as a result they began cloning, often breaking off bigger chunks than they needed. The increased architecture size and complexity itself forced the hiring of more engineers, creating a vicious cycle. The DLC group, however, hired at a slower pace, and it introduced all new engineers to the architecture with an informal apprenticeship.

Managing risks and opportunities

The DLC group managed technical and market risks in a number of ways. At specific stages, for example, it reviewed the architecture with internal and external customers and stakeholders, tracking and testing the assumptions underlying customer requirements. One manager described these reviews this way: "We routinely decide to *buy* more information. The amount of information we buy depends on the amount of risk we

Architecture is most effective in delivering value and managing complexity when the six critical organizational principles are applied in concert.

perceive.” The process of *buying* information included white-board meetings with component owners, multiple cycles of quick prototyping, and if necessary, management or customer involvement.

After one review, a manager suspected that a planned enhancement to an existing component had in it hidden complexity. He stopped development and ordered the component’s behavior analyzed using a finite-state-machine model. The model showed that the enhancement was several times more complicated than originally envisioned, and the manager was able to rework the enhancement and avoid major integration problems.

The DLC group also regularly prioritized risks, focusing on the high-risk areas first. Risky areas were tested with prototypes representing alternative technical approaches, and the chosen solution was the first to be implemented, tested, and integrated. To avoid disrupting the rhythm of releases, the group delivered high-risk features in phases, over multiple releases. Nortel reported that few architecture-related problems made it through this process and that the company was above the industry average for defect removal.

With the growing investment by both public- and private-sector organizations in software product-line architecture, managers need to know the organizational factors affecting its success. Nortel—a leading telecommunication firm with large-scale, mission-critical experience—has stretched the limits of software architecture, reaped extraordinary success, and learned from its inevitable mistakes. What it has learned and how it is now applying those lessons hold important insights for management.

Our advisers helped us develop six critical organizational principles, and our study of Nortel confirmed each of them. For Nortel, not adequately applying them has had serious consequences, but when the company has applied the principles, the results have been very positive. We observed, however, that applying a principle without discrimination—too much, too little, or without focus—also caused negative consequences. How does one strike a balance? Our conclusion is that architecture is most effective in delivering value and managing complexity when the six critical organizational principles are applied in concert.

We believe the richness of this approach comes from learning how the principles interact across a variety of organizations. Applied Expertise Inc., of Arlington, Virginia, is now developing benchmarking techniques to measure how and with what effect organizations apply these principles. Applied Expertise is using these techniques to benchmark several companies that are recognized leaders in the application of product-line architecture, including AT&T, Hewlett-Packard, and Nortel. The techniques hold additional promise for application in the fields of software reuse and patterns. ❖

Acknowledgments

We thank Nortel for allowing us to conduct the case study and for hosting the advisers’ workshop. We also thank our sponsors, the Defense Advanced Research Projects Agency’s STARS program and Office of Secretary of Defense for funding the case study and contributing to the production of this article.

We also acknowledge the contributions made by the study’s advisers: Grady Booch, David J. Bristow, Robert Charette, Patricia Cornwell, Glenn Dillard, Tom Goodall, Ron Grace, Kwan Tse Huang, Conni Marchewka, Mike Pait, Rick Peebles, Bob Savely, Robert Sonneman, Bill Spotz, John Willison, and John Woodfin.

We also thank Rock Angier, Francois Coallier, Michael Deutsch, Deborah Hamilton, Geoffrey Moore, James W. Moore, David Preston, Stan Rifkin, David Rine, Bill Saunders, and Allen Stennett for their reviews, suggestions, and comments. Study team members Marcia Carlyn and Carol Terry developed and validated the research protocol. And program managers John Foreman and Linda Brown helped shape and guide this project for DARPA and OSD.

References

1. D. Garlan and D. Perry, “Introduction to the Special Issue on Software Architecture,” *IEEE Trans. on Software Eng.*, Apr. 1995, pp. 269–274.
2. B. Ziegler, “What Really Happened at Northern Telecom,” *Business Week*, Aug. 9, 1993, pp. 27–28.
3. M. Griss, J. Favaro, and P. Walton, “Managerial and Organizational Issues: Starting and Running a Software Reuse Program,” *Software Reusability,* W. Schaefer, R. Prieto-Diaz, and M. Matsumoto, eds., Ellis Horwood, New York, 1994.
4. F. Fafchamps, “Organizational Factors and Reuse,” *IEEE Software*, Sept. 1994, pp. 31–41.
5. C. Morris and C. Ferguson, “How Architecture Wins Technology Wars,” *Harvard Business Rev.*, Mar./Apr. 1993, pp. 86–96.
6. GAO, *Case Study Evaluations,* Transfer Paper 10.1.9, Governmental Printing Office, Washington, D.C., Nov. 1990.
7. G. Booch, *Object Solutions: Managing the Object-Oriented Project,* Addison Wesley, Reading, Mass., 1995, p. 29.
8. K. Weick, *The Social Psychology of Organizing,* Addison Wesley, Reading, Mass., 1979, pp. 188–193.
9. P. Cashin, “BNR Remains at Forefront of Computing Technology,” *Telesis,* July 1991, pp. 18–19, 74.
10. R. Cooper, “Debunking the Myths of New Product Development,” *Research Technology Management*, July/Aug. 1994, pp. 40–50.

Dave Dikel is the vice president and director of research for Applied Expertise Inc. He is interested in capturing and transferring engineering experience using patterns and benchmarking, particularly in the areas of software architecture and reuse. Dikel received a BA in philosophy and physical science from the University of California, San Diego. He is a member of IEEE, ACM, ASQC, and the IEEE Software Engineering Standards Committee's Reuse Steering Committee.

David Kane is a project executive and software engineer at Applied Expertise Inc. His research interests include patterns, management of risk, and the Internet. Kane is currently enrolled at George Mason University in the MS program for software systems engineering; he received a BS in computer science and mathematics from Binghamton University. He is a member of IEEE, ACM, and ASQC.

Steve Ornburn is an adviser for R&D effectiveness at Nortel. He is interested in software engineering methods, including reuse, reverse engineering, and component generators. He received an MS in computer science from the Georgia Institute of Technology and a BS in industrial engineering from Northwestern University. He is a member of IEEE and ACM.

William Loftus is president and technical director of WPL Laboratories Inc. His research interests include real-time and distributed systems, simulation, intranet applications, and program generation. He received a BS and an MS in computer science from Villanova University. He is a member of IEEE and ACM.

Jim Wilson is president of Applied Expertise Inc. and a software engineer. His research focuses on benchmarking, software reuse, and Web databases. He received a BA in American studies and physics from Bucknell University and an MA in science and technology policy from George Washington University.

Contact Dikel at Applied Expertise Inc., 1925 North Lynn Street, Suite 802, Arlington, VA 22209; ddikel@aecorp.com. For more information on Applied Expertise Inc., see http://www.aecorp.com.

High Performance Computing '98

Grand Challenges in Computer Simulation

Part of the 1998 Advanced Simulation Technologies Conference (ASTC'98)

April 5–9, 1998 • Boston Park Plaza Hotel • Boston, Massachusetts

The 1998 Advanced Simulation Technologies Conference will feature the Sixth Special Symposium devoted to the impact of high performance computing on computer simulation.

Topics to be addressed include:

- High Performance Computing Applications
- High Performance Communication Networks
- Methodologies to Exploit HPC Architectures
- Parallel Algorithms and Numerical Methods
- Visualization and Validation of Large-Scale Computer Simulations

The goal of the Symposium is to encourage innovation in high performance computing technologies and to stimulate the use of these technologies in key areas of computer simulation. It promotes the exchange of ideas and information among universities, industry, and national laboratories supporting the development of high performance computing applications, which can provide solutions to current scientific and technical challenges.

Applications of interest include: (but are not limited to)

- 3-D Computational Fluid Dynamics Models
- Prediction of Weather, Climate, and Global Changes
- Electrical Power Generation and Distribution
- Computational Chemistry
- Automotive Applications
- Telecommunication Applications and Networking
- Combustion Systems Simulation
- Computational Solid Mechanics Applications
- Computational Ocean Sciences
- Life Sciences and Medical Applications
- Intelligent Transportation Systems Simulation
- Petri Nets Applications

Prospective authors are invited to submit abstracts (400–900 words) describing their work in these areas, by September 26, 1997. E-mail submissions accepted. Expected conference registration fee is $380^{00}, which includes publication in the HPC'98 proceedings.

Send Abstracts to:
HPC'98 • Dr. Adrian Tentner, Chair
c/o SCS • P.O. Box 17900 • San Diego, CA 92177
e-mail: tentner@anl.gov

Additional information can be obtained from:
SCS • P.O. Box 17900 • San Diego, CA 92177
tel: (619) 277-3888 • fax: (619) 277-3930
e-mail: scs@scs.org • http://www.scs.org/confernc/Astc98/

Sponsored by The Society for Computer Simulation International

SCS